

R workflow for tree segmentation from ALS data

Jean-Matthieu Monnet

2021-02-01

The code below presents a tree segmentation workflow from Airborne Laser Scanning (lidar remote sensing) data. The workflow is based on functions from R packages `lidaRtRee` and `lidR`, and it includes the following steps:

- treetop detection,
- crown segmentation,
- accuracy assessment with field inventory,
- species classification

Steps 1 and 3 are documented in (Monnet et al. 2010; Monnet 2011). The detection performance of this algorithm was evaluated in a benchmark (Eysn et al. 2015).

Licence: CC-BY / source page

Material

Field inventory

The field inventory corresponds to a 50m x 50m plot located in the Chablais mountain (France) (Monnet 2011, 34). All trees with a diameter at breast height above 7.5 cm are inventoried. The data is available in package `lidaRtRee`.

```
# load dataset from package (default)
data(treeinventorychablais3, package = "lidaRtRee")
```

Otherwise you can load your own data provided positions and heights are measured.

```
# import field inventory
fichier <- "chablais3_listeR.csv"
tree.inventory <- read.csv(file = fichier, sep = ";", header = F)
names(tree.inventory) <- c("x", "y", "d", "h", "n", "s", "e", "t")
# save as rda for later access
# save(tree.inventory, file="tree.inventory.rda")
```

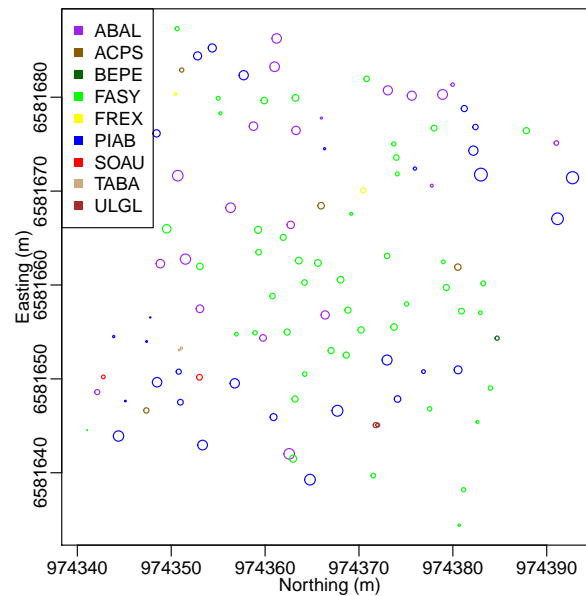
Attributes are:

- x easting coordinate in Lambert 93
- y northing coordinate in Lambert 93
- d diameter at breast height (cm)
- h tree height (m)
- n tree number
- s species abbreviated as GESP (GENus SPecies)
- e appearance (0: missing or lying, 1: normal, 2: broken treetop, 3: dead with branches, 4: snag)
- t tilted (0: no, 1: yes)

```
##           x           y           d           h n           s e t
## 1 974353.3 6581643 37.6 23.6 1 PIAB 1 0
## 2 974351.0 6581648 15.7 13.9 2 PIAB 1 0
## 3 974348.5 6581650 34.2 23.0 3 PIAB 1 0
```

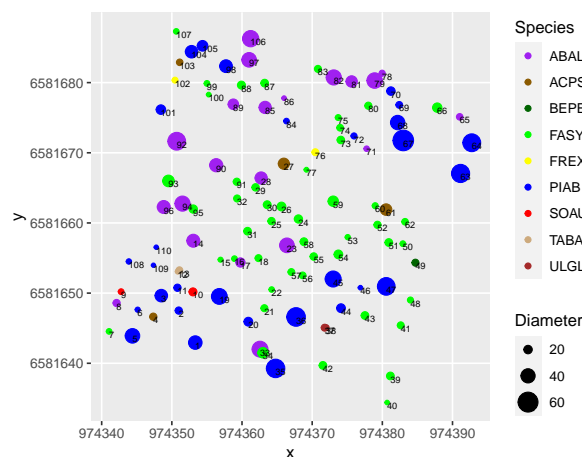
Function `plotTreeInventory` is designed to plot forest inventory data.

```
# display inventoried trees
lidaRtRee::plotTreeInventory(
  treeinventorychablais3[, c("x", "y")],
  treeinventorychablais3$h,
  species = as.character(treeinventorychablais3$s)
)
```



The ggplot2 package also provides nice outputs.

```
# use table of species of package lidaRtRee to always use the same color for a given species
plot.species <- lidaRtRee::speciesColor()[levels(treeinventorychablais3$s), "col"]
library(ggplot2)
ggplot(treeinventorychablais3, aes(x = x, y = y, group = s)) +
  geom_point(aes(color = s, size = d)) +
  coord_sf(datum = 2154) +
  scale_color_manual(values = plot.species) +
  scale_radius(name="Diameter") +
  geom_text(aes(label=n, size=20), hjust=0, vjust=1) +
  labs(color = "Species") # titre de la légende
```



We define the region of interest (ROI) to crop ALS data to corresponding extent before further processing. ROI is set on the extent of tree inventory, plus a 10 meter buffer.

```
# buffer to apply around ROI (meters)
ROI.buffer <- 10
# ROI limits
ROI.range <- data.frame(round(apply(treeinventorychablais3[,c("x","y")],2,range)))
```

Airborne Laser Scanning data

In this tutorial, ALS data available in the ‘lidaRtRee’ package is used.

```
# load data in package lidaRtRee (default)
data(laschablais3, package="lidaRtRee")
laschablais3

## class      : LAS (v1.2 format 1)
## memory     : 7 Mb
## extent     : 974326, 974408, 6581619, 6581702 (xmin, xmax, ymin, ymax)
## coord. ref. : RGF93 / Lambert-93
## area       : 6803.003 m²
## points     : 92.1 thousand points
## density    : 13.54 points/m²
```

Otherwise, ALS data is loaded with functions of package `lidR`. First a catalog of files containing ALS data is built. Then points located inside our ROI are loaded.

```
# directory for laz files
lazdir <- "/directory_of_las_files/"
# build catalog of files
cata <- lidR::readLAScatalog(lazdir)
# set coordinate system
sp::proj4string(cata) <- sp::CRS(SRS_string = "EPSG:2154")
# set sensor type
lidR::sensor(cata) <- "ALS"
# extract points in ROI plus additional buffer
laschablais3 <- lidR::clip_rectangle(cata,
                                     ROI.range$x[1]-ROI.buffer-5,
                                     ROI.range$y[1]-ROI.buffer-5,
                                     ROI.range$x[2]+ROI.buffer+5,
                                     ROI.range$y[2]+ROI.buffer+5)

# save as rda for easier access:
# save(laschablais3, file="laschablais3.rda", compress = "bzip2")
```

Data preparation

Digital Elevation Models

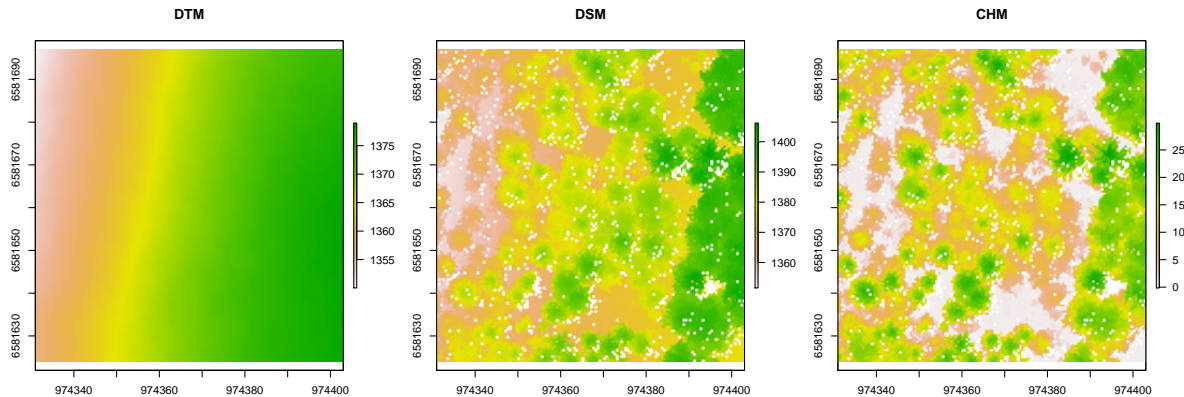
From the ALS point cloud, digital elevation models are computed (Monnet 2011, 43–46). The Digital Terrain Model (DTM) represents the ground surface, it is computed by bilinear interpolation of points classified as ground. The Digital Surface Model (DSM) represents the canopy surface, it is computed by retaining in each raster’s pixel the value of the highest point included in that pixel. The Canopy Height Model (CHM) is the normalized height of the canopy. It is computed by subtracting the DTM to the DSM.

```
# terrain model computed from points classified as ground
dtm <- lidaRtRee::points2DTM(lidR::filter_ground(laschablais3),
                             res=0.5, ROI.range$x[1]-ROI.buffer,
                             ROI.range$x[2]+ROI.buffer, ROI.range$y[1]-ROI.buffer,
                             ROI.range$y[2]+ROI.buffer)

# surface model
```

```
dsm <- lidaRtRee::points2DSM(laschablais3, res=0.5, dtm@extent@xmin,
                             dtm@extent@xmax, dtm@extent@ymin, dtm@extent@ymax)

# canopy height model
chm <- dsm - dtm
```



Visual comparison of field inventory and ALS data

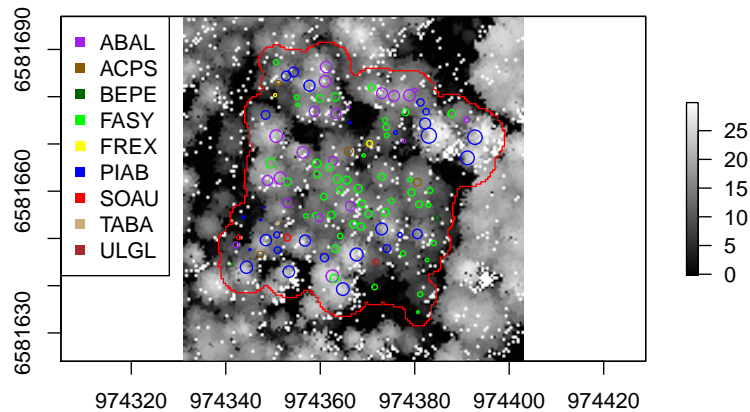
A plot mask is computed from the inventoried positions, using a height-dependent buffer. Indeed, tree tops are not necessarily located vertically above the trunk positions. In order to compare detected tree tops with inventoried trunks, buffers have to be applied to trunk positions to account for the non-verticality of trees.

```
# select trees with height measures
selec <- which(!is.na(treeinventorychablais3$h))
# plot mask computation based on inventoried positions
# convex hull of plot
ChullMask <- lidaRtRee::rasterChullMask(treeinventorychablais3[selec, c("x", "y")], dsm)
# union of buffers around trees
TreeMask <- lidaRtRee::rasterXYMask(treeinventorychablais3[selec, c("x", "y")],
                                   2.1+0.14*treeinventorychablais3$h[selec], dsm)
# union of convexHull and tree buffers
plotMask <- max(ChullMask, TreeMask)
# vectorize plot mask
v.plotMask <- raster::rasterToPolygons(plotMask, function(x) (x==1), dissolve=T)
```

Displaying inventoried trees on the CHM shows a pretty good correspondance of crowns visible in the CHM with trunk locations and sizes.

```
# display CHM
raster::plot(chm, col=gray(seq(0,1,1/255)),
             main="Canopy Height Model and tree positions")
# add inventoried trees
lidaRtRee::plotTreeInventory(treeinventorychablais3[, c("x", "y")],
                             treeinventorychablais3$h,
                             species=as.character(treeinventorychablais3$s), add=TRUE)
# display plot mask
raster::plot(v.plotMask, border="red", add=TRUE)
```

Canopy Height Model and tree positions



Tree delineation

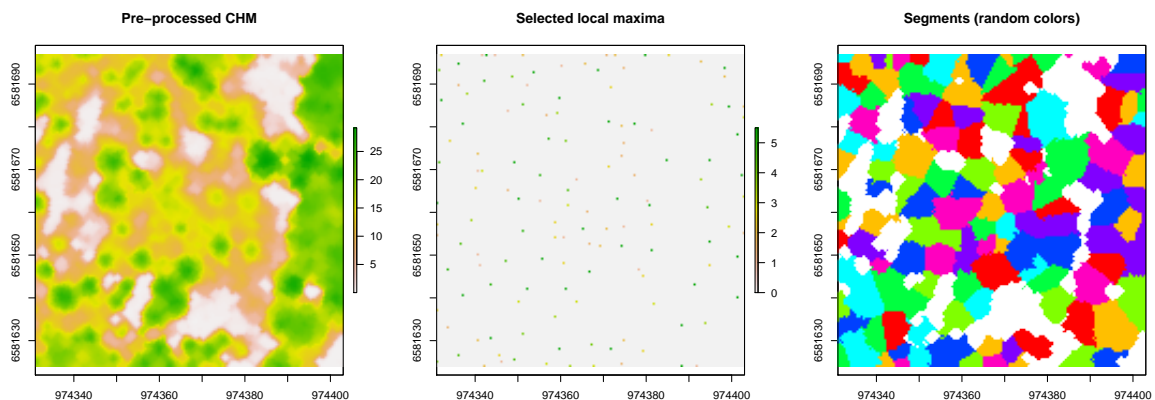
Tree segmentation

Tree segmentation is performed on the Canopy Height Model by using a general function which consists in the following steps:

- image pre-processing (non-linear filtering and smoothing for noise removal)
- local maxima filtering and selection for tree top detection
- image segmentation with a watershed algorithm for tree delineation.

The first two steps are documented in Monnet (2011), pp. 47-52.

```
# tree detection (default settings), applied on canopy height model
segms <- lidaRtRee::treeSegmentation(chm)
#
par(mfrow=c(1,3))
# display pre-processed chm
raster::plot(segms$smoothed.dem, main="Pre-processed CHM")
# display selected local maxima
raster::plot(segms$local.maxima, main="Selected local maxima")
# display segments, except ground segment
dummy <- segms$segments.id
dummy[dummy==0] <- NA
raster::plot(dummy %% 8, main="Segments (random colors)", col=rainbow(8), legend=FALSE)
```



Tree extraction

A data.frame of detected trees located in the plot mask is then extracted with the function `treeExtraction`. Segments can be converted from raster to polygons but the operation is quite slow. Attributes are :

- id: tree id
- x: easting coordinate of tree top
- y: northing coordinate of tree top
- h: height of tree top
- dom.radius: distance of tree top to nearest crown of neighbouring tree with larger height
- s: crown surface
- v: crown volume
- sp: crown surface inside plot
- vp: crown volume inside plot

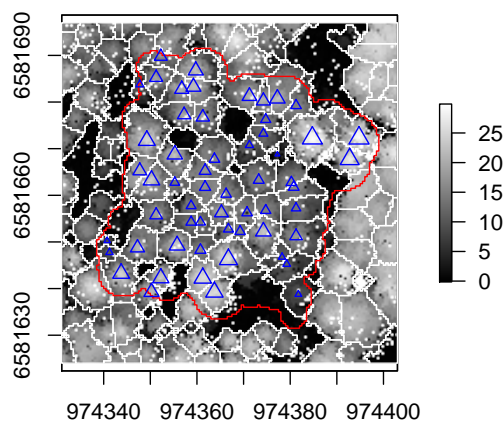
```
# tree extraction only inside plot mask for subsequent comparison
trees <- lidaRtRee::treeExtraction(segms$filled.dem,
                                   segms$local.maxima,
                                   segms$segments.id, plotMask)

head(trees, n=3L)
```

```
##   id      h dom.radius    s      v    sp      vp
## 1  4 10.65143      1.0 20.25 189.6750 20.25 189.6750
## 2  5 18.54525      5.5 61.50 697.1207 61.00 694.2223
## 3  8 20.75848      2.0 17.75 289.3803 17.75 289.3803
```

```
# convert segments from raster to polygons
v.segments <- raster::rasterToPolygons(segms[[2]], dissolve=T)
#
# display initial image
raster::plot(chm, col=gray(seq(0,1,1/255)), main="CHM and detected positions")
# display segments border
sp::plot(v.segments, border="white", add=T)
# display plot mask
sp::plot(v.plotMask, border="red", add=T)
# display inventoried trees
graphics::points(trees$x, trees$y, col="blue", cex=trees$h/20, pch = 2)
```

CHM and detected positions



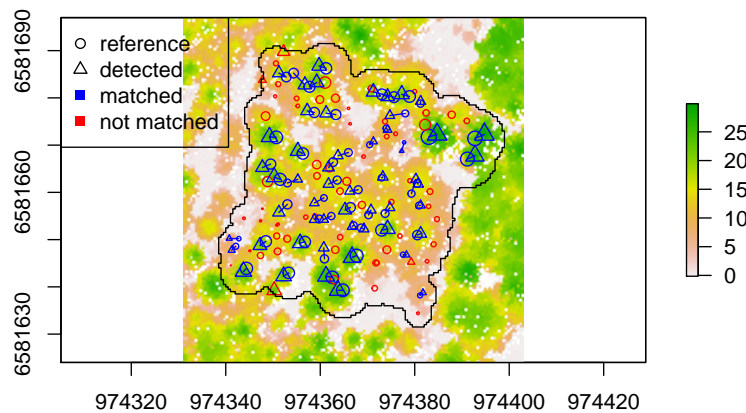
Detection evaluation

Tree matching

To assess detection accuracy, reference (field) trees should be linked to detected trees. Despite the possibility of error, automated matching has the advantage of making the comparison of results from different algorithms and settings reproducible and objective. The algorithm presented below is based on the 3D distance between detected treetops and inventory positions and heights (Monnet et al. 2010).

```
# match detected treetops with field trees based on relative distance of apices
matched <- lidaRtRee::treeMatching(treeinventorychablais3[selec,c("x","y","h")],
                                   cbind(trees@coords, trees$h))

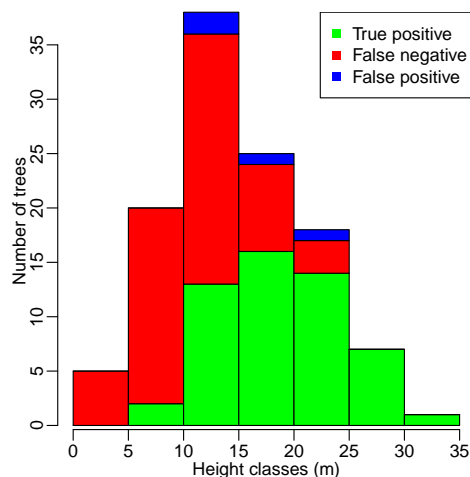
# display matching results
lidaRtRee::plot2Dmatched(treeinventorychablais3[selec,c("x","y","h")],
                         cbind(trees@coords, trees$h), matched, chm, v.plotMask)
```



Detection accuracy

Detection accuracy is evaluated thanks to the number of correct detections (53), false detections (4) and omissions (57). In heterogeneous stands, detection accuracy is height-dependent, it is informative to display those categories on a height histogram.

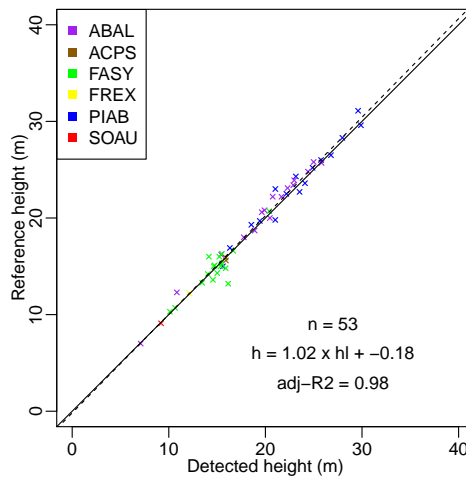
```
# height histogram of detections
detection.stats <- lidaRtRee::histDetection(treeinventorychablais3[selec,c("x","y","h")],
                                             cbind(trees@coords, trees$h), matched)
```



Height estimation accuracy

For true detections, estimated heights can be compared to field-measured heights. Here, the root mean square error is 1.5m, while the bias is -0.2m. The linear regression is displayed hereafter.

```
# linear regression between reference height and estimated height
heightReg <- lidaRtRee::heightRegression(treeinventorychablais3[selec,c("x","y","h")],
                                         cbind(trees@coords, trees$h),
                                         matched, species=treeinventorychablais3$s)
```



Species Classification

Points in segments

Before computation of point cloud metrics in each segment, the whole point cloud is normalized to convert point altitude to height above ground. Points are then labeled with the id of the segment they belong to. A list of LAS objects corresponding to the segments is then created.

```
# normalize point cloud
lasn <- lidR::normalize_height(laschablais3, lidR::tin())
# add segment id in LAS object
lasn@data$seg.id <- raster::extract(segms[["segments.id"]], lasn@data[,1:2])
# split las object by segment id
lasl <- split(lasn@data, lasn@data$seg.id)
# convert list of data.frames to list of las objects
lasl <- lapply(lasl, function(x){lidR::LAS(x, lasn@header)})
# set coordinate system
dummy <- sp::CRS(SRS_string = "EPSG:2154")
for (i in 1:length(lasl)) {sp::proj4string(lasl[[i]]) <- dummy}
```

Metrics computation

Basic point cloud metrics are computed in each segment (maximum, mean, standard deviation of heights, mean and standard deviation of intensity).

```
# compute basic las metrics in each segment
metrics <- lidaRtRee::cloudMetrics(lasl, func=~list(maxZ=max(Z), meanZ=mean(Z),
                                                    sdZ=sd(Z), meanI=mean(Intensity),
                                                    sdI=sd(Intensity)))

# create segment id attribute
metrics$seg.id <- row.names(metrics)
head(metrics, n=3L)
```

```
##      maxZ      meanZ      sdZ      meanI      sdI seg.id
## 0   8.68   0.6081085 1.3053197 84.72108 61.62381      0
```



```
## 1 18.71 16.7692188 0.9809055 61.62500 47.22842      1
## 2 23.93 11.0996272 7.1666867 69.16710 61.74219      2
```

Merge with reference and detected trees

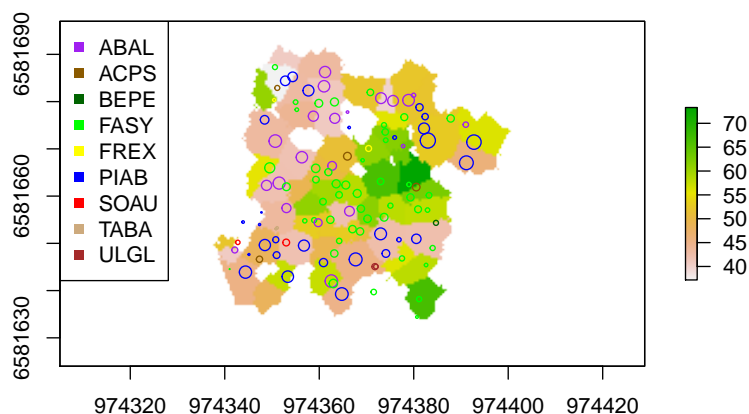
Computed metrics are merged with reference and detected trees, thanks to the segment id.

```
# associate each reference tree with the segment that contains its trunk.
treeinventorychablais3$seg.id <- raster::extract(segms[["segments.id"]],
                                                treeinventorychablais3[,c("x","y")])
# create new data.frame by merging metrics and inventoried trees based on segment id
tree.metrics <- base::merge(treeinventorychablais3,metrics)
# remove non-tree segment
tree.metrics <- tree.metrics[tree.metrics$seg.id!=0,]
# add metrics to extracted tree data.frame
trees <- base::merge(trees, metrics, by.x="id", by.y="seg.id", all.x=T)
```

Plotting the reference trees with the mean intensity of lidar points in the segments shows that when they are dominant, broadleaf trees tend to have higher mean intensity than coniferous trees. .

```
# create raster of segment' mean intensity
r.mean.intensity.segm <- segms[["segments.id"]]
# match segment id with id in metrics data.frame
dummy <- match(raster::values(r.mean.intensity.segm), trees$id)
# replace segment id with corresponding mean intensity
raster::values(r.mean.intensity.segm) <- trees$meanI[dummy]
# display tree inventory with mean intensity in segment background
raster::plot(r.mean.intensity.segm, main="Mean intensity in segment")
# display tree inventory
lidaRtRee::plotTreeInventory(treeinventorychablais3[,c("x","y")],
                             treeinventorychablais3$h,
                             species=as.character(treeinventorychablais3$s), add=T)
```

Mean intensity in segment



Exploratory analysis

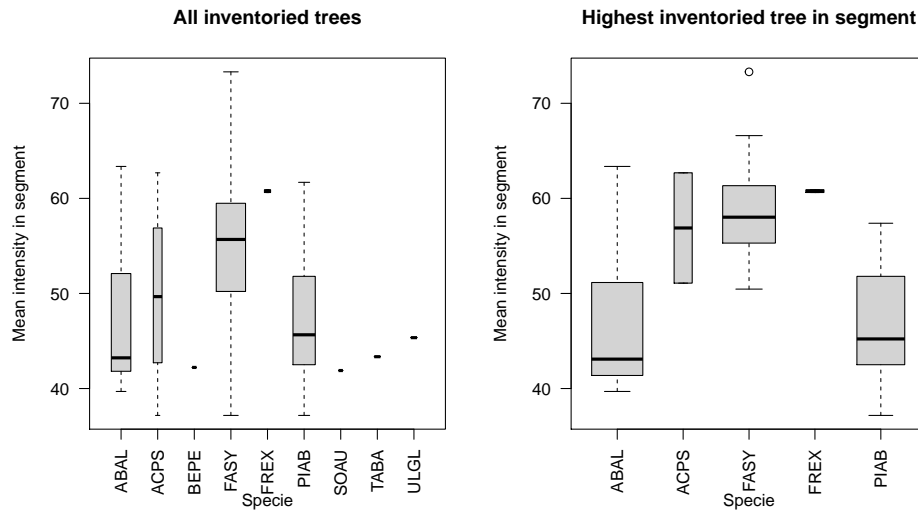
A boxplot of mean intensity in segments per species shows that mean intensity distribution is different between species. The analysis can be run by considering only the highest inventoried trees in each segment, as smaller trees are likely to be suppressed and have smaller contribution to the point cloud.

```
par(mfrow=c(1,2))
boxplot(meanI~s,data=tree.metrics[,c("s","maxZ", "meanZ", "sdZ", "meanI", "sdI")],
```

```

ylab="Mean intensity in segment", xlab="Specie",
main="All inventoried trees", las=2, varwidth = TRUE)
boxplot(meanI~s,data=tree.metrics.h, ylab="Mean intensity in segment",
xlab="Specie", main="Highest inventoried tree in segment", las=2,
varwidth = TRUE)

```

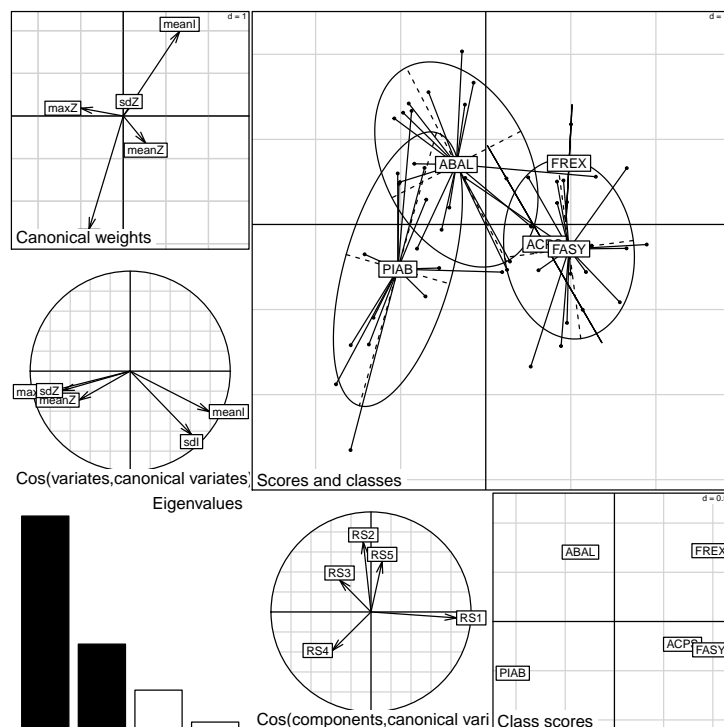


A linear discriminant analysis shows that main species can be discriminated, thanks to a combination of height and intensity variables.

```

acp2 <- ade4::dudi.pca(tree.metrics.h[,c("maxZ", "meanZ", "sdZ", "meanI", "sdI")],scannf=F)
afd <- ade4::discrim(acp2,tree.metrics.h$s,scannf=F, nf=2)
plot(afd)

```



Display point cloud

The point cloud can be displayed colored by segment, with poles at the location of inventoried trees.

```

rgl::par3d(mouseMode = "trackball") # parameters for interaction with mouse
# select segment points and offset them to avoid truncated coordinates in 3d plot
points.seg <- lasn@data[which(lasn@data$seg.id!=0), c("X", "Y", "Z", "seg.id")]
points.seg$X <- points.seg$X - 974300
points.seg$Y <- points.seg$Y - 6581600
# plot point cloud
rgl::plot3d(points.seg[,c("X", "Y", "Z")], col=points.seg$seg.id%%10 +1,aspect=FALSE)
#
# add inventoried trees
treeinventorychablais3$z <- 0
for (i in 1:nrow(treeinventorychablais3))
{
  rgl::lines3d(rbind(treeinventorychablais3$x[i]-974300,
                    treeinventorychablais3$x[i]-974300),
              rbind(treeinventorychablais3$y[i]-6581600,
                    treeinventorychablais3$y[i]-6581600),
              rbind(treeinventorychablais3$z[i],
                    treeinventorychablais3$z[i]+treeinventorychablais3$h[i]))
}

```

/tmp/Rtmpcs2wIR/file2c13d2650cd36.png

Batch processing

The following code exemplifies how to process numerous LAS files and extract trees for the whole area with parallel processing. The processing runs faster if data is provided as chunks to the segmentation algorithm, and results are then aggregated, rather than running on the full coverage of the data. In order to avoid border effects, chunks are provided to the algorithm as overlapping tiles. Results are cropped to prevent the same tree from appearing twice in the final results. Tile size and buffer size are important parameters :

- tile size is a trade-off between the number of chunks to process and the amount of RAM required to process a single tile ;
- buffer size is a trade-off between redundant processing of the overlap area, and assuring that a tree is which treetop is located at the border of a tile has its full crown within the buffer size.

Tiles can be processed with parallel computing, within limits of the cluster's RAM and number of cores.

The steps for processing a batch of las/laz files are :

- build catalog of files
- provide tiling parameters
- provide segmentation parameters
- provide output parameters
- set cluster options for parallel processing
- compute the X and Y coordinates of tiles
- parallelize the processing with the built-in package `parallel`, by sending to the clusters the coordinates of tiles to process, and a function with the instructions to proceed :
 - load tile corresponding to the coordinates
 - compute CHM
 - segment and extract trees
- aggregate list of results

Be aware that in case tree segments are vectorized, some obtained polygons might overlap. The segmentation algorithm might not be deterministic and borders are sometimes not consistent when adjacent polygons are pasted from different tiles.

```

rm(list=ls())
# BUILD CATALOG OF FILES
# folder containing the files
lazdir <- "./data/forest.structure.metrics"

```

```

# build catalog
cata <- lidR::catalog(lazdir)
# set coordinate system
sp::proj4string(cata) <- sp::CRS(SRS_string = "EPSG:2154")

## Warning in showSRID(SRS_string, format = "PROJ", multiline = "NO", prefer_proj =
## prefer_proj): Discarded datum Réseau Geodesique Francais 1993 in CRS definition

# set sensor type
lidR::sensor(cata) <- "ALS"
#
# BATCH PROCESSING PARAMETERS
# tile size to split area into chunks to process
# trade-off between RAM capacity VS total number of tiles to process
tile.size <- 70 # here 70 for example purpose with small area
# buffer size: around each tile to avoid border effects on segmentation results
# trade-off between making sure a whole tree crown is processed in case its top
# is on the border VS duplicate processing
buffer.size <- 10 # 5 m is minimum, 10 is probably better depending on tree size
#
# TREE SEGMENTATION PARAMETERS
# set raster resolution
resolution <- 1
#
# OUTPUT PARAMETERS
# option to vectorize tree crowns (set to FALSE if too slow)
vectorize.trees <- TRUE
# output canopy height models ? (set to FALSE if too much RAM used)
output.chm <- TRUE
# save chms on disk
save.chm <- FALSE
#
# CLUSTER PARAMETERS
# working directory
wd <- getwd()
# run with two cores
clust <- parallel::makeCluster(getOption("cl.cores", 2))
#, outfile = "/home/jean-matthieu/Bureau/output.txt")
# export global variables to cluster because they will be called inside the function
parallel::clusterExport(cl = clust, ls(), envir = .GlobalEnv)
#
# COORDINATES OF TILES
# low left corner
x <- seq(
  from = floor(cata@bbox["x", "min"] / tile.size) * tile.size,
  by = tile.size,
  to = ceiling(cata@bbox["x", "max"] / tile.size - 1) * tile.size
)#[1:2]
length.x <- length(x)
y <- seq(
  from = floor(cata@bbox["y", "min"] / tile.size) * tile.size,
  by = tile.size,
  to = ceiling(cata@bbox["y", "max"] / tile.size - 1) * tile.size
)#[1:2]
length.y <- length(y)
# repeat coordinates for tiling while area
x <- as.list(rep(x, length.y))
y <- as.list(rep(y, each = length.x))

```

```

#
# PARALLEL PROCESSING
# function takes coordinates of tile as arguments
resultats <- parallel::mcmapply(
  # i and j are the coordinated of the low left corner of the tile to process
  FUN = function(i, j)
  {
    # set working directory
    setwd(wd)
    # initialize output
    output <- list()
    output$name <- paste0(i, "_", j)
    # extract tile plus buffer
    las.tile <- lidR::clip_rectangle(
      cata,
      i - buffer.size,
      j - buffer.size,
      i + tile.size + buffer.size,
      j + tile.size + buffer.size
    )
    # check if points are present
    if (nrow(las.tile@data)>0)
    {
      # normalization if required
      # las.tile <- lidR::normalize_height(las.tile, lidR::tin())
      # in this example LAS tiles are already normalized
      # compute canopy height model
      chm <- lidaRtRee::points2DSM(las.tile)
      # check all chm is not NA
      if (!all(is.na(raster::values(chm))))
      {
        # output chm if asked for
        if (output$chm) output$chm <- raster::crop(chm, raster::extent(i, i+tile.size,
                                                                           j, j+tile.size))

        # save on disk
        if (save$chm) raster::writeRaster(raster::crop(chm,
                                                         raster::extent(i, i+tile.size,
                                                                           j, j+tile.size)),
                                           file=paste0("chm_", i, "_", j, ".tif"),
                                           overwrite = TRUE)

        #
        # tree detection
        segms <- lidaRtRee::treeSegmentation(chm)
        # tree extraction
        trees <- lidaRtRee::treeExtraction(segms$filled.dem,
                                           segms$local.maxima,
                                           segms$segments.id)

        # remove trees in buffer area
        trees <- trees[trees$x >= i & trees$x < i + tile.size &
                      trees$y >= j & trees$y < j + tile.size, ]
        # add tile id to trees to avoid duplicates in final file
        trees$tile <- paste0(i,"_",j)
        # convert to vectors if option is TRUE
        if (vectorize$trees)
        {
          # vectorize
          v.trees <- raster::rasterToPolygons(segms$segments.id, dissolve = T)
          # remove polygons which treetop is in buffer

```

```

    v.trees <- v.trees[is.element(v.trees$segments.id, trees$id), ]
    # names(v.trees) <- "id"
    # add attributes
    # errors when using sp::merge so using sp::over even if it is probably slower
    # merge(v.trees@data, trees, all.x = TRUE)
    v.trees@data <- cbind(v.trees@data, sp::over(v.trees, trees))
    # save in list
    output$v.trees <- v.trees
  }
  # save trees in list
  output$trees <- trees
} # end of raster is not all NAs check
} # end of nrow LAS check
output
}, x, y, SIMPLIFY = FALSE) # function applied to the lists of coordinates (x and y)
parallel::stopCluster(cl = clust)
#
# RESULTS AGGREGATION
# extract results from nested list into separate lists and then bind data
id <- unlist(lapply(resultats, function(x) x[["name"]]))
#
# trees
trees <- lapply(resultats, function(x) x[["trees"]])
# remove NULL elements
trees[sapply(trees, is.null)] <- NULL
# bind remaining elements
trees <- do.call(rbind, trees)
#
# chm
if (output$chm)
{
  chm <- lapply(resultats, function(x) x[["chm"]])
  # merge chm
  # no names in list otherwise do.call returns an error
  chm.all <- do.call(raster::merge, chm)
  names(chm) <- id
}
# v.trees
if (vectorize.trees)
{
  v.trees <- lapply(resultats, function(x) x[["v.trees"]])
  # remove NULL elements
  v.trees[sapply(v.trees, is.null)] <- NULL
  v.trees <- do.call(rbind, v.trees)
  # 1-pixel overlapping in v.trees might be present because image segmentation
  # is not fully identical in overlap areas of adjacent tiles.
}

```

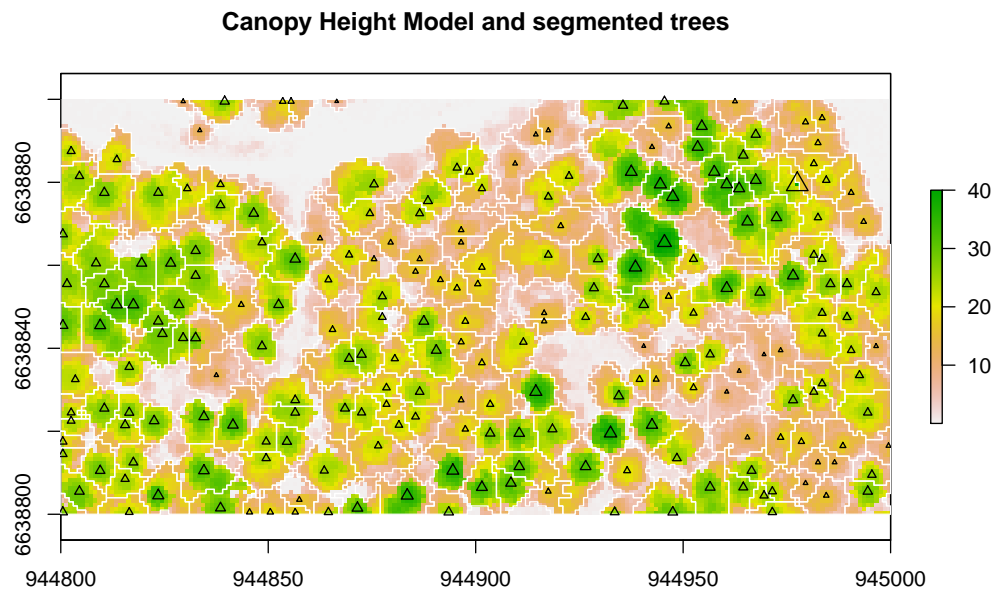
The following image displays the results for the whole area.

```

# threshold outsiders in chm
chm.all[chm.all > 40] <- 40
chm.all[chm.all < 0] <- 0
# display chm
raster::plot(chm.all,
             main = "Canopy Height Model and segmented trees")
# display segments border
sp::plot(v.trees, border = "white", add = T)

```

```
# add trees
sp::plot(trees, cex = trees$h/40, add = TRUE, pch = 2)
```



The following lines save outputs to files.

```
# merged chm
raster::writeRaster(chm.all, file= "chm.tif")
#trees
raster::shapefile(trees, file="trees.points.shp")
#vectorized trees
if (vectorize.trees) raster::shapefile(v.trees, file="trees.polygons.shp")
```

References

- Eysn, Lothar, Markus Hollaus, Eva Lindberg, Frédéric Berger, Jean-Matthieu Monnet, Michele Dalponte, Milan Kobal, et al. 2015. "A Benchmark of Lidar-Based Single Tree Detection Methods Using Heterogeneous Forest Data from the Alpine Space." *Forests* 6 (12): 1721–47. <https://doi.org/10.3390/f6051721>.
- Monnet, Jean-Matthieu. 2011. "Using Airborne Laser Scanning for Mountain Forests Mapping: Support Vector Regression for Stand Parameters Estimation and Unsupervised Training for Treetop Detection." PhD thesis, Université de Grenoble. <http://tel.archives-ouvertes.fr/tel-00652698/fr/>.
- Monnet, Jean-Matthieu, Eric Mermin, Jocelyn Chanussot, and Frédéric Berger. 2010. "Tree top detection using local maxima filtering: a parameter sensitivity analysis." In *10th International Conference on LiDAR Applications for Assessing Forest Ecosystems (Silvilaser 2010)*, 9 p. Freiburg, Germany. <https://hal.archives-ouvertes.fr/hal-00523245>.